User Guide

JWare/AntXtras Svn4Ant

Release v1.1.0

Sandbox Software MC
1133 Broadway, Suite 706
New York, NY. 10010
http://purl.net/jware/

Simone Cato

Version 1.1.0, Revised Feb 2008

Table of Contents

Introduction

This document is the official user reference of the JWare/AntXtras Svn4Ant package (Svn4Ant); it gives you a comprehensive description, including examples, of all Svn4Ant ***script*** components. This document does not contain any extended build script tutorials; nor does it describe how you can extend Svn4Ant to create your own Ant components. For this information you should go to the AntXtras website (purl.net/jware/). You will also find the Svn4Ant javadocs on the website. Each component's javadoc contains additional examples and implementation details important to developers.

Most of this guide assumes you have installed the Svn4Ant distribution and have read the Quick Start pages on the website to become familiar with the basic Svn4Ant capabilities.

Terms of Use

Svn4Ant Quick Start Review

The JWare/AntXtras Svn4Ant (Svn4Ant) components are divided into two main categories ***Client*** and ***Admin***. The client components comprise the bulk of the items described in this document. Immodestly, Svn4Ant's goal is to be the best quality provider for Subversion Ant and Maven components.

How to Read Descriptions

Each Svn4Ant element's description contains its default Svn4Ant antlib name, the category (admin or client), a brief description, the list of most-used parameters, the list of nestable elements (if any), a set of examples, the list of related items (if any), and optionally a list of known issues. Although each item's description is labeled using our conventions, the full class name is also supplied so you can redefine the Ant script element however you want.

Installation

The JWare / AntXtras Svn4Ant (Svn4Ant) installation is similar to that of any optional Ant package. The following instructions describe how to install and verify Svn4Ant in your Ant runtime environment.

1. If you haven't already done so, download and install an Ant distribution—at least version 1.6.5 or 1.7.0. Verify that Ant is properly installed by trying a simple Ant script.

2. Download, verify, and extract an Svn4Ant distribution. For the first-time user we suggest you download the "-withdeps" binary distribution that includes all the required third-party libraries that Svn4Ant needs— you can use the sample "`svn4ant-install-check.xml`" script to verify the installation is working properly.

   F If you must manually generate all binaries for your environment, download the source-only distribution. The included "`ez-build.xml`" Ant build file can generate a default distribution from the source. Be sure you configure the "`ez-build.properties`" file for your environment. You must use JDK version 1.4 or later to build and generate the Svn4Ant package using the `ez-build.xml` Ant file.

   In the remaining steps we will use `<SVN4ANT_DIR>` to refer to the directory into which the Svn4Ant distribution was extracted or built.

3. Update your Ant runtime environment to include the Svn4Ant jar file `<SVN4ANT_DIR>/lib/svn4ant.jar` and all of its dependencies in your Ant classpath. There are several ways of telling Ant about third-party jar files; the easiest method is to copy the jar files into your Ant distribution's `lib` directory. A safer approach is to install Svn4Ant in its own location and update the `CLASSPATH` used when you run Ant (for example, by using the `–lib` option to run Ant or a custom `$HOME/.antrc` file).

4. Verify the Svn4Ant tasks are accessible from Ant. The easiest way to do this is to run Ant against one of the sample or test build files from the distribution. From within the `<SVN4ANT_DIR>/etc` directory, run '`ant –f svn4ant-install-check.xml`'. This sample build file loads the Svn4Ant antlib. If Ant is unable to locate the Svn4Ant classes, even this simple build file will fail.

5. Read the rest of this guide for an overview of the most useful Svn4Ant tasks.

6. Using the install script as a guide, include the Svn4Ant antlib in your build file(s) using a standard `<taskdef>`.

7. Start using Svn4Ant!

Common Parameters

Common Client and Admin Parameters

This section defines several parameters common to all Svn4Ant tasks, client and admin. We describe these parameters here to eliminate the need to duplicate their descriptions in every task. Note that these parameters are an addition to the parameters that all Ant components inherit; for example, parameters like "`description`" and "`taskname`".

Attribute	Description	Required
`feedback`	Set to level of feedback of (diagnostic) output. Possible values are "`none`", "`normal`", "`verbose`", and "`quiet`". Note that the amount of feedback that is "`normal`" differs from task to task.	No; defaults "`normal`"
`haltiferror`	Set to "`no`" if the task should not signal a build error if it cannot execute ***or cannot execute completely***. Note that this option does not cover malformed or missing parameters; these conditions will still signal a build error. Possible values are any acceptable synonym for `true` and `false`.	No; defaults "`yes`"
`failproperty`	Set to the name of a property to set if this task is unable to execute successfully for any reason. This property is set to the error's message regardless of the `haltiferror` parameter's setting.	No
`failvariable`	Set to the name of a variable to set if this task is unable to execute successfully for any reason. This variable is set to the error's message regardless of the `haltiferror` parameter's setting.	No

Common Client Parameters

This section defines several parameters common to most Svn4Ant ***client*** tasks. We describe these parameters here to eliminate the need to duplicate their descriptions in every client task that supports them. Any exceptions (tasks that do not support a particular parameter) are noted in the rightmost "Except" column.

Attribute	Description	Required	Except
`username`	Set to the user name to present for authentication to the Subversion repository; use in conjunction with `password`. Matches the standard `svn` client `--username` option.	No; only one type of auth when used.	`<svn>`
`password`	Set to the password to present for authentication to the Subversion repository; use in conjunction with `username`. Matches the standard `svn` client `--password` option.		`<svn>`
`credential`	Set to a reference id of either a `<svnserver>`, `<svnserverset>`, or `<svncredential>` data object. Svn4Ant will extract authentication, proxy, caching and other repository information (depending		

Attribute	Description	Required	Except
	on task) from the named target.		
revisionproperty	Set to the name of a property to update with any repository revision number(s). For some tasks, Svn4Ant will not set this property if the underlying Subversion library does not return new revision information.	No	<svn>
authcache	Set to "no" to prevent Svn4Ant from caching authentication credentials. Matches the standard svn client --no-authcache option.	No	<svn>
configdir	Set to the location of a local configuration directory from which Subversion library should read its "config" options. Matches the standard svn client --config-dir option.	No	<svn>

Client Components

This section defines the client-side Svn4Ant components. You will need to install the SVNKit library and its dependencies into your Ant classpath before you can use these components. Read the Svn4Ant installation instructions for additional details.

High-Speed Client Component Overview

Define your Subversion credentials for re-use by all Svn4Ant tasks:

```
<svnserver id="my.repo" isdefault="yes" authcache="no">
    <url value="http://svn.example.com/"/>
    <credential username="myuserid" password="mypasswd"/>
</svnserver>
```

Create a new working copy of trunk@HEAD:

```
<svncheckout from="myproject/trunk" to="${work.dir}"/>
```

Update an existing working copy:

```
<svnupdate path="${work.dir}"/>
```

Get the revision a working copy represents for use in a build label:

```
<svnrevget path="${work.dir}" property="build.reporev"/>
```

Commit a changed working copy:

```
<svncommit path="${work.dir}">
    <message>The reason for changes…</message>
</svncommit>
```

Create a snapshot (tag) of trunk@HEAD:

```
<svnbranch externals="pin"
   from="myproject/trunk" to="myproject/tags/${snapshot.name}"
   message="Snapshot for myproject (${snapshot.name})"/>
```

Export a local working copy with modifications as a standalone tarred and gzipped archive:

```
<svnexport from="${work.dir}" to="${ftp.dir}/website-${BSTAMP}+tar+gzip"
    addlocals="yes" externals="yes"/>
```

Create a new “standard” Subversion project with this Ant macrodef `<mksvnproject>`:

```
<macrodef name="mksvnproject">
    <attribute name="name"/>
    <attribute name="repo" default="my.repo"/>
    <sequential>
        <svn action="mkdir" credential="@{repo}">
            <argument value="${$svnurl:@{repo}}/@{name}"/>
            <argument value="${$svnurl:@{repo}}/@{name}/trunk"/>
            <argument value="${$svnurl:@{repo}}/@{name}/tags"/>
            <argument value="${$svnurl:@{repo}}/@{name}/branches"/>
            <argument line="-m ‘Initial setup’"/>
        </svn>
    </sequential>
</macrodef>
```

SvnCredential <svncredential>

Class: com.idaremedia.svn4ant.clientauth.SvnCredential **Category:** Client (Data)

Description

The SvnCredential type (defined `<svncredential>`) lets you define reusable repository-independent authentication information like a username and password or a keyfile and passphrase. You can also include reusable connection information like proxy and port settings if, for example, you always connect to a repository through a company firewall. You would use an `<svncredential>` to keep your connection information in one place in a script instead of scattered throughout multiple commands.

Although the contents of an `<svncredential>` is not specific to a particular Subversion repository (see `<svnserver>`), you are limited to a single login credential per item. You cannot, for instance, define both a username / password login and a keyfile / passphrase login in a single `<svncredential>` object.

You can define each bit of credential information as either a shorthand parameter or as a nested element (whose name is the same as the attribute name)— not both. The field's value is defined as the element's text or as the value of a single "`value`" parameter; see the Examples section below.

Parameters

Attribute	Description	Required
`username`	User name used to connect and authenticate to the Subversion repository.	Yes.
`keyfile`	Path to your SSH key file. If a relative path is set, it will be resolved relative to the ant project's base directory. This information will be used if a "`+ssh`" protocol is used to connect to a respository (e.g. "`svn+ssh://`").	No
`password`	Password used to authenticate to the Subversion repository; use in conjunction with `username` or `keyfile`.	Yes; one of these is required (even if blank).
`passphrase`	Passphrase used to authenticate to the Subversion repository; use in conjunction with `username` or `keyfile`.	
`sshport`	Port number used when connecting to the SSH server. This information will be used if a "+ssh" protocol is used to connect to a respository (e.g. "`svn+ssh://`"). Ignored if `defaultconfig` is enabled.	No
`proxy`	URL of proxy server used for connecting to the Subversion server instances. Ignored if `defaultconfig` is enabled.	No
`proxyport`	Port number of the named `proxy` server. Ignored unless a proxy server is defined and `defaultconfig` is not enabled.	No
`refid`	Reference to another `<svncredential>` object. If defined no other parameter (except the inherited "`id`") can be defined.	No

Nested Elements

Each of the credential-related parameters can be defined as a standalone nested element of the same name; for example, the `passphrase` parameter can be defined as a nested `<passphrase>` element whose value is to be the element's character text or the value of a single "`value`" attribute.

Examples

The following snippet declares a general anonymous login that also goes through a company web proxy:

```
<svncredential id="login.public">
    <username>anonymous</username>
    <password>anonymous</password>
    <proxy>webproxy.mycompany.com</proxy>
    <proxyport>8000</proxyport>
</svncredential>
```

The following snippet declares the same anonymous login but uses shorthand parameters instead of nested elements to declare all the information:

```
<svncredential id="login.public"
    username="anonymous" password="anonymous"
    proxy="webproxy.mycompany.com" proxyport="8000"/>
```

The following snippet declares an SSH-based credential for a user "`jack`". This snippet also shows the two different ways you can define a nested credential field's value:

```
<svncredential id="login.jack" username="jack">
    <passphrase>DEaTH to *any* who _pilfers_ my Lunch!</passphrase>
    <keyfile value="/home/jack/.buildssh/.ssh_rsa"/>
</svncredential>
```

The following snippet declares a credential that uses the AntX `$password:` value URI handler to extract password information from an external file instead of the build file itself (a more secure solution if your build scripts are distributed):

```
<svncredential id="login.nightlybuild" username="nightlybuild"
   passphrase="${$password:nightlybuild}"/>
```

See Also

- The `<svnserver>` item lets you define additional connection information for a specific Subversion repository.

SvnServerDef <svnserver>

Class: com.idaremedia.svn4ant.clientauth.SvnServerDef **Category:** Client (Data)

Description

The SvnServerDef type (defined `<svnserver>`) lets you define a reusable connection including credential to a particular Subversion repository. You would use an `<svnserver>` to keep your connection information to a single repository in one place in a script instead of scattered throughout multiple commands. You can also mark a server definition as the ***default*** repository connection for a project and Svn4Ant will use it for all client commands that do not include their own repository and credential information.

If you want Svn4Ant credential to supplement the regular credential caching that Subversion clients do (as controlled by your global Subversion configuration), you should set the "`defaultconfig`" parameter to "`yes`". This will let Svn4Ant know that it should add the local Ant configuration to the established lookup mechanism instead of replacing that mechanism.

Parameters

Attribute	Description	Required
`defaultconfig`	Set to "`yes`" to have Svn4Ant supplement the normal Subversion client credential lookup mechanism instead of replacing it. Enabling this option lets Svn4Ant take advantage of existing global credential configuration and caches.	No; defaults "`no`"
`authcache`	Set to "`yes`" or "`no`" to explicitly define a credential caching policy for all components that use this definition. Overrides both client-side and server configuration.	No; defaults to Subversion default
`isdefault`	Set to "`yes`" to use this definition as the default repository connection for all Svn4Ant client commands. Only used if the command does not include its own credential parameter.	No; defaults "`no`".
`refid`	Reference to another `<svnserver>` object. If defined, no other parameter (except the inherited "`id`") can be defined.	No

Nested Element: `<credential>`

The `<credential>` element defines the authentication credential for the repository. Its format is the same as the standalone `<svncredential>` data item. You can reference a standalone credential by setting this item's "`refid`" parameter to that credential's `id`.

Nested Element: `<realm>`

An optional element, the `<realm>` element defines the server definition's security realm. This definition will only present its credentials if the target repository's security realm is a match to this value. When you define a realm, the server definition never uses its URL to determine if it should present its credential.

Nested Element: `<url>`

The `<url>` element defines the repository's URL. This value does not have to point to the repository root; it can point into the repository to a particular subdirectory. The URL value is the item's text or the "`value`" parameter like: `<url value="http://localhost/repos"/>`. If you have not defined a realm, the server definition will use its URL to determine whether it should present its credential; for a match, this URL must be the (grand)parent of the requested URL.

Nested Element: `<configdir>`

An optional element, the `<configdir>` element defines a custom directory for your Subversion configuration; this information is not used by Svn4Ant directly but by the underlying SVNKit library. You define the directory's path as the item's text or as the "`value`" parameter like: `<configdir value="C:\builds\subversion\conf"/>`.

Nested Element: `<directory>`

An optional element, the `<directory>` element defines the local directory of a repository. You can define a local location if your build script and repository reside on the same file system. You define the directory's path as the item's text or as the value of the "`value`" parameter like: `<directory value="C:\exported\subversion\qa"/>`. Currently, only the Svn4Ant `$svndir:` value uri handler uses this information.

Examples

The following snippet declares a default server definition for a company's "`softdev`" repository. Unless explicitly overwritten on each command, every Svn4Ant ***client*** command will use this definition's information. Note that the credential "`nightlybuild`" has been defined elsewhere and it will not be cached by the underlying Subversion client library:

```
<svnserver id="repo.softdev" isdefault="yes" authcache="no">
    <credential refid="login.nightlybuild"/>
    <url value="svn://softdev/applications"/>
</svnserver>
```

The following snippet declares a server definition for a specific location ***inside*** a "`vendors`" repository. The first time this definition is used from a command the credential might be cached depending on the global `auth-cache` option and the contents of the `config` file in the custom configuration directory `${buildconf}/repos/vendors`:

```
<svnserver id="repo.vendors">
    <url value="https://svn.libs.org/java/stable"/>
    <credential>
        <username>mycompany</username>
        <passphrase>Some Super-Seekrit Words Here</passphrase>
    </credential>
    <configdir value="${buildconf}/repos/vendors"/>
</svnserver>
```

See Also

- The `<svncredential>` type lets you define repository-independent login information.
- The `$svnurl:` value URI lets you extract an svnserver's URL as a dynamic value.

SvnServerSet <svnserverset>

Class: com.idaremedia.svn4ant.clientauth.SvnServerSet **Category:** Client (Data)

Description

The SvnServerSet type (defined `<svnserverset>`) lets you define a collection of credentials for different repositories. You would use an `<svnserverset>` with Svn4Ant components that can touch multiple repositories each with its own credential requirement; for example, the `<svnupdate>` task might need to read information from different repositories routinely.

On a non-Windows platform, Svn4Ant (via SVNKit) can read the cached credentials generated by other Subversion clients lessening the need to define a set of credentials within your scripts. However, on Windows, 'wincrypt'-ed credentials are unreadable by pure Java-based applications like Svn4Ant so you must define your credentials in a form the underlying SVNKit library can understand. Additionally, the cached credentials (readable or not) would usually belong to a logged in user; this may not be the same user your Ant scripts would use to perform its repository functions. (Note that as of SVNKit 1.1.4 you can read wincrypt-ed credentials from Windows but you need to use a platform specific SVNKit library in order to do it.)

Parameters

Attribute	Description	Required
`defaultconfig`	Set to "`yes`" to have Svn4Ant supplement the normal Subversion client credential lookup mechanism instead of replacing it. Enabling this option lets Svn4Ant take advantage of existing global credential configuration and caches.	No; defaults "`no`"
`defaultserver`	Set to the id of the `<svnserver>` this set considers its default. The default server's settings will be used for relative repository URLs and other cases where a single `<svnserver>`'s information is needed.	No
`refid`	Reference to another `<svnserverset>` object. If defined, no other parameter (except the inherited "`id`") can be defined.	No

Nested Element: `<server>`

The `<server>` element lets you define a single `<svnserver>` definition directly in the server set. You can either define the item completely inline or just refer to an existing `<svnserver>` object. See the description of the `<svnserver>` type for more information.

Examples

The following snippet defines a set of credentials based on pre-existing standalone `<svnserver>` items. The server set will also use the local Subversion credential caches if the current runtime is not a Windows based system:

```
<svnserverset id="allrepos" defaultconfig="${$not:isWindows}">
   <server refid="repo.tests"/>
   <server refid="repo.javalibraries"/>
   <server refid="repo.licenses"/>
</svnserverset>
```

The following snippet is similar to the one above but the server definitions are done inline to the server set itself. Also this server set defines a default credential explicitly (`repo.tests`) so the script can use the server set as a complete replacement for a simpler `<svnserver>` credential:

```
<svnserverset id="allrepos" defaultserver="repo.tests">
   <server id="repo.tests" authcache="no">
      <credential refid="nightlybuild"/>
      <realm value="SVN Lab"/>
      <url value="http://svn.lab/repos/rw/programmertests"/>
   </server>
   <server id="repo.javalibraries">
      <credential username="mavenuid" password="${$password:mavenuid}"/>
      <url value="http://svn.my.com/maven"/>
   </server>
</svnserverset>
```

See Also

- The `<svnserver>` type lets you define a single-repository credential if that's all you need.

SvnLibCheckTask <svn-libcheck>

Class: com.idaremedia.svn4ant.client.info.SvnLibCheckTask **Category:** Client (Diagnostics)

Description

The SvnLibCheckTask task (defined `<svn-libcheck>`) is a diagnostics task that lets you determine the active versions of Svn4Ant and TMate's SVNKit. Calling `<svn-libcheck>` generates two properties: one for Svn4Ant (`svn4ant.label`) and another for SVNKit (`svnkit.label`). After using the task, your Ant script can display or evaluate the values of these properties.

Parameters

Attribute	Description	Required
`prefix`	A prefix to prepend to the builtin property names.	No

Nested Elements

The `<svn-libcheck>` does not support any nested elements.

Examples

This snippet displays the current Svn4Ant and SVNKit labels which includes version and other product information:

```
<svn-libcheck/>
<echo message="Svn4Ant: ${svn4ant.label}, SVNKit: ${svnkit.label}"/>
```

This snippet extracts and displays the same information except the properties are prefixed with an underscore "_":

```
<svn-libcheck prefix="_"/>
<echo message="Svn4Ant: ${_svn4ant.label}, SVNKit: ${_svnkit.label}"/>
```

See Also

- The `<svnadmin-libcheck>` performs a similar function for the native Subversion Java bindings library, `libsvnjavahl` that Svn4Ant uses to implement its repository admin tasks.
- The `<vendorinfo>` AntX task lets you extract very detailed information about the loaded Svn4Ant ; for example, you could extract just the version number to use in a condition check.

SvnCliTask <svn>

Class: com.idaremedia.svn4ant.client.jsvn.SvnCliTask **Category:** Client

Description

The SvnCliTask task (defined `<svn>`) is the Svn4Ant wrapper around the standard SVNKit Subversion client. Bascially, this task lets you call the SVNKit client from within your Ant scripts using accepted Antisque nomenclature. Anything you can do with the SVNKit client, you can do with `<svn>`. Conversely, things you ***cannot*** do from the SVNKit client, you still cannot do from the `<svn>` task; you will need to use the Svn4Ant custom tasks.

Note: because Svn4Ant does not provide an independent task for every Subversion client sub-command, you will need to use `<svn>` to perform these operations from within Ant. Some functions in particular like 'mkdir', 'status', and 'info' require that you use the `<svn>` task.

Svn4Ant does extend the standard SVNKit client to support our `<svncredential>` item through a `credential` parameter; this allows you to specify SSH-based credentials and proxy connection information without relying on the various system properties that the standalone SVNKit client has to. And while the other common Svn4Ant parameters: `feedback`, `haltiferror`, and `failproperty` are also supported, the SVNKit client, because it expects to be run as a standalone console application, has a habit of exiting the running JVM when it encounters an error; this will abort the Ant runtime unconditionally, regardless of the `haltiferror` and `failproperty` settings.

Parameters

Attribute	Description	Required
`credential`	Reference id to either a `<svnserver>` or `<svncredential>` data object. Svn4Ant will extract authentication, proxy, caching and other repository information (depending on svn subcommand) from the named target.	No
`action`	Set to the name of the Subversion client subcommand to perform. The name must be the command's "long name" or one of its recognized aliases (as per the official Subversion client accepted aliases). ***Do not include this value as a nested argument.***	Yes
`args`	Convenient shortcut for a single line parameter. Equivalent to a single nested `<argument line="…"/>` element.	No
`outputfile`	Path to the file where standard output from the SVNKit client should be saved.	No
`errorsfile`	Path to the file where standard error from the SVNKit client should be saved.	No
`append`	Set to "`yes`" if the command output should be appended to either or both output files. (Covers both types of files.)	No

Nested Element: `<argument>`

The `<argument>` element lets you define the input command line to the SVNKit client; its format is the same as the standard Ant command line `<argument>` element. See the Ant documentation for further information.

Examples

This antlib snippet uses the `<svn>` task to create a set of remote directories in the repository defined by the “`svnrepo`” server definition. The snippet also shows how you can use the Svn4Ant `$svnurl:` value URI handler to extract information from a server definition.

```
<macrodef name="mksvnproject">
    <attribute name="name"/>
    <attribute name="repo" default="svnrepo"/>
    <sequential>
        <assert isref="@{repo}" msg="‘@{repo}’ is defined as reference"/>
        <svn action="mkdir" credential="@{repo}">
            <argument value="${$svnurl:@{repo}}/@{name}"/>
            <argument value="${$svnurl:@{repo}}/@{name}/trunk"/>
            <argument value="${$svnurl:@{repo}}/@{name}/tags"/>
            <argument value="${$svnurl:@{repo}}/@{name}/branches"/>
        </svn>
    </sequential>
</macrodef>
```

This snippet uses the `<svn>` task to get and save information on a repository defined on the command line itself. The information is saved to the “`svnstat.out`” file. Svn4Ant will read the repository credential information from the default `<svnserver>` definition:

```
<svn action="info" outputfile="${buildlogs}/svnstat.out">
    <argument value="${basedir}"/>
</svn>
```

This snippet uses the `<svn>` task to capture the last log message commited to a repository. All of the command options including credentials are defined as nested arguments using all of the options available through this standard Ant element:

```
<svn action="propget">
  <argument line="--username me --password seekrit --no-auth-cache"/>
  <argument line="svn:log --revprop"/>
  <argument value="http://svn.lab/repo/theproject"/>
  <argument line="-r HEAD"/>
</svn>
```

This snippet uses the `<svn>` task to setup the externals properties on a project’s `lib` directory. All of the command options except credentials are defined as nested arguments using all of the options available through this standard Ant element:

```
<svn action="propset" credential="svnrepo">
    <argument value="svn:externals"/>
    <argument value="-F"/>
    <argument file="${project.home}/lib/.defs"/>
    <argument file="${project.home}/lib"/>
</svn>
<svn action="ci" credential="svnrepo"
     args="-m LIB-EXTERNALS ${project.home}"/>
```

SvnImportTask <svnimport>

Class: com.idaremedia.svn4ant.client.solo.SvnImportTask **Category:** Client

Description

The SvnImportTask task (defined `<svnimport>`) lets you import a local directory tree or distribution archive into a repository. You can use Ant's standard `defaultexcludes` option as well as a Subversion-specific `defaultignores` option to omit or include certain files and directories before import. And, unlike the standard Subversion client, you can direct `<svnimport>` to immediately checkout a newly imported directory in place. Note that archives are extracted into a temporary directory that is automatically deleted once the task completes (even if the import is unsuccessful).

Parameters

The following parameters are ***in addition*** to the common parameters inherited by all Svn4Ant client tasks. Read the "Common Client Parameters" section for more information.

Attribute	Description	Required
`from`	Path to local directory or archive to be imported.	Yes
`to`	Path to repository directory ***into*** which local directory's contents imported. The named repository must exist, the specific path within the repository will be created if necessary.	Yes; is relative to any svnserver URL
`recurse`	Set to "`no`" to only import the top-level directory's contents. Unlike the nested `<delete>` item; this option does not affect the local directory's contents before import.	No; defaults "`yes`"
`defaultexcludes`	Set to "`yes`" to have Svn4Ant automatically remove items matching the current Ant default exclusion list. These items are ***deleted*** from the local directory before the import.	No; defaults "`no`".
`defaultignores`	Set to "`no`" to have Svn4Ant block the default ignored file list and import such files (and directories).	No; defaults "`yes`".
`message`	Message associated with this import.	No; only one when used. Defaults to the empty string.
`messagefile`	Path to a local file containing the message associated with this import. This file must exist.	
`encoding`	Set to the encoding of the message file if it is different from platform default. Ignored unless `messagefile` is defined.	No
`checkout`	Set to "`yes`" to have Svn4Ant automatically checkout the imported directory into the same local directory from which it was imported. ***This will erase the directory's contents completely before attempting the checkout.***	No; defaults "`no`".
`strict`	Set to "`yes`" to block all feedback and have the import task echo only the new revision id of the repository.	No

Nested Element: <message>

The `<message>` element lets you specify a more complex import message than the shorthand message parameter.

Nested Element: <delete>

The `<delete>` element is a standard Ant `<patternset>` that lets you specify a filter for the imported files and directories; items excluded by the filter ***will be removed*** from the local directory before it is imported. Note that this patternset is an addition to the set embodied by the `defaultexcludes` option. The format of the the `<delete>` element is exactly that of the standard Ant `<patternset>`; read the Ant documentation for further information on pattern sets.

Examples

This snippet imports the contents of the local directory "`${newproj}`" into the repository defined by the property "`${repo.url}`". If authentication is required, Svn4Ant will look for a default `<svnserver>` definition for information. Once the import is complete, Svn4Ant will immediately create a working copy in the original import directory by doing a checkout from the new remote area:

```
<mksvnproject name="myproj"/>
<svnimport from="${newproj}" to="${repo.url}/myproj/trunk" checkout="yes"/>
```

This snippet imports the contents of a local gzipped tarball in "`${downloads}`" into the repository defined by "`repo.vendors`" server definiton and the subpath "`${archive}`". The repository revision associated with the import is saved in the "REV" property:

```
<svnimport credential="repo.vendors"
  from="${downloads}/${archive}.tgz" to="${archive}"
  message="Stable release ${version}" revisionproperty="REV"/>
```

This snippet imports the contents of a local vendor release "`${release}`" into the repository location defined by the "`repo.vendors`" server definition and the "`${vendor}/${release}`" subpath. Before the import, all default excludes as well as items matching a specific pattern are removed:

```
<svnimport credential="repo.vendors" defaultexcludes="yes"
    from="${downloads}/${release}" to="${vendor}/${release}">
    <delete>
        <include name="**/lib/*"/>
        <include name="**/website/"/>
    </delete>
</svnimport>
```

See Also

- The `<svnexport>` task lets you export a repository directory tree to a local package.
- The `<svncopy>` task lets to copy one repository or working copy location to another repository location.

Class: com.idaremedia.svn4ant.client.solo.SvnCheckoutTask **Category:** Client

Description

The SvnCheckoutTask task (defined `<svncheckout>`) lets you create a local working copy of a repository directory's contents. Currently you can checkout directories only; Subversion does not support single file checkouts (see `<svncat>`).

You can use a single `<svncheckout>` command to fetch a set of working copies in one operation. The working copies are checked out to a shared parent directory (itself not under version control).

Parameters

The following parameters are ***in addition*** to the common parameters inherited by all Svn4Ant client tasks. Read the "Common Client Parameters" section for more information.

Attribute	Description	Required
`from`	Path to repository directory on which the local working copy is based. This repository location must exist.	Yes unless a nested `<item>` is used.
`to`	Path to a local directory; Svn4Ant will create this directory if necessary. You cannot point to an existing versioned directory unless the directory is rooted at the same repository as this checkout, or you enable the `clean` option to remove the local files. If you define nested `<item>` elements, Svn4Ant uses this `to` parameter as the parent (unversioned) directory into which all the working copies are checked out.	Yes unless a ***single*** nested `<item>` is used.
`revision`	Set to the revision you want checked out. If '`to`' is not the root of a working copy and '`revision`' is different from the BASE revision, only the tree beneath '`to`' is updated.	No; defaults "`HEAD`"
`clean`	Set to "`yes`" to have the checkout erase ***all*** the contents of an existing local directory before the checkout operation.	No; default "`no`"
`recurse`	Set to "`no`" to block a recursive checkout. If recursion is blocked, only the ***files*** within the named repository directory are checked out (no subdirectories are checked out).	No; defaults "`yes`"
`force`	Set to "`yes`" to force this operation to refetch a working copy's file list from the server to ensure all missing or new items are retrieved. Use if the working copy was previously updated with the `recurse` option disabled.	No; defaults "`no`"
`externals`	Set to "`no`" to disable checkout of items defined by the special "`svn:externals`" directory property.	No; defaults "`yes`"

Nested Element: `<item>`

The `<item>` element lets you specify a single checkout instruction that occurs as part of a set of different checkouts under a single parent (unversioned) directory. You would use `<item>` elements if your workspace is comprised of multiple independent project directories. If you do not define an item's '`to`' parameter, Svn4Ant will use the base name of the checked out directory.

Parameters

Attribute	Description	Required
`from`	Path to repository directory on which the working copy is based. This repository location must exist.	Yes
`to`	Path to a local directory; Svn4Ant will create this directory if necessary. If this is a relative path, it is created relative to the parent `<svncheckout>` task's '`to`' parameter.	No; defaults to basename of directory
`revision`	Set to the revision you want checked out. This parameter has the same limitations as the '`revision`' option of the parent task.	No; defaults "`HEAD`"

Nested Element: `<items>`

The `<items>` element lets you reuse a set of checkout instructions defined elsewhere as an `<svntransferset>`. Because you can load an `<svntransferset>` from a file or other external source, the `<items>` element lets you alter the list of directories to checkout independent of the main script file. Read the description of the `<svntransferset>` item for more information.

Examples

The following snippet checks out a working copy of the trunk subdirectory in the repository location defined by the "`devbox.repo`" credential:

```
<svncheckout credential="devbox.repo" from="trunk" to="${workdir}"/>
```

The following snippet checks out just the top-level files of a project's trunk subdirectory at the specific repository revision `123`.

```
<svncheckout username="anonymous" password="anonymous"
    from="${repo}/proj/trunk" to="${tmpdir}" revision="123"
    recurse="no"/>
```

The following snippet checks out two independent versions of separate repository locations under a single parent directory:

```
<svncheckout to="${dependencies.dir}" credential="login.anon">
   <item from="http://svn.libs.net/svnkit/nightly/jars" to="svnkit/lib"/>
   <item from="svn://localhost/javalibraries/antxtras/nightly" to="antxtras"/>
</svncheckout>
```

See Also

- The `<svnupdate>` task lets you synchronize a working copy with repository.
- The `<svncommit>` task lets you commit local modifications to the repository.

- The `<svncat>` task lets you retrieve the contents of a single file to a local copy.

SvnUpdateTask <svnupdate>

Class: com.idaremedia.svn4ant.client.solo.SvnUpdateTask **Category:** Client

Description

The SvnUpdateTask task (defined `<svnupdate>`) lets you update one or more versioned items to a specific repository revision. To update multiple independent working copies that share a common parent directory, you can use the `search` parameter. Set the value of the `search` parameter to the maximum number of subdirectory levels the operation should search for working copy roots. The operation will update each root it finds using the parameters you have defined on the `<svnupdate>` instance.

The `<svnupdate>` task has a builtin fileset filter that it can apply to the set of possible files it can update. To filter the items Svn4Ant updates, you can use either a nested `<items>` patternset element, or any of the common `<include>`, `<exclude>` file selectors.

Parameters

The following parameters are ***in addition*** to the common parameters inherited by all Svn4Ant client tasks. Read the "Common Client Parameters" section for more information.

Attribute	Description	Required
`path`	Path to a local filesystem object. If the `search` option is disabled, this filesystem object (a file or directory) must be under Subversion control. If the `search` parameter is enabled, this option should be the path to an existing directory.	Yes
`search`	Set to the number of levels that Svn4Ant should decend looking for independent working copy roots. If set to "`*`" Svn4Ant will look for all working copy roots below the directory specified in the `path` parameter.	No; defaults "`0`"
`revision`	Set to the revision you want the local copies updated to. If `path` is not the root of a working copy and `revision` is different from the current BASE revision, only the tree beneath `path` is updated.	No; defaults "`HEAD`"
`recurse`	Set to "`no`" to block a recursive update. If recursion is blocked, only the named `path` ***item*** is updated. For directories—no directory contents are updated.	No; defaults "`yes`"
`force`	Set to "`yes`" to force this operation to refetch a working copy's file list from the server to ensure all missing or new items are retrieved. Use if the working copy was previously updated with the `recurse` option disabled.	No; defaults "`no`"
`leaveconflicts`	Set to "`yes`" to leave update (merge) conflicts unresolved without touching the local file contents. ***Note that the conflict will still exist***; this just prevents SVNKit from munging the local file with ">>>>>>" markers for external merge tools.	No; defaults "`no`"

Attribute	Description	Required
externals	Set to “no” to disable update of items defined by the special “svn:externals” directory property.	No; defaults “yes”

Nested Element: <items>

The <items> element is a standard Ant <patternset> that lets you specify a filter for the updated files and directories; ***only items included by the filter will be updated***. The format of the the <items> element is exactly that of the standard Ant <patternset>; read the Ant documentation for further information on pattern sets.

Nested Element: <include>

The <include> element lets you quickly define a file name selector for the update task’s builtin patternset. The format is the exactly that of the standard Ant patternset <include> element.

Nested Element: <exclude>

The <exclude> element lets you quickly define an exclusion file name selector for the update task’s builtin patternset. The format is the exactly that of the standard Ant patternset <exclude> element.

Examples

The following snippet updates a working copy to the latest revision in its repository. The “devbox.repo” credential is used to respond to any repository authentication challenge:

```
<svnupdate credential="devbox.repo" path="${workdir}"/>
```

The following snippet updates all the working copies located under the directory defined by the “${modules}” property. No svn:externals defined items are updated.

```
<svnupdate path="${modules.dir}" force="yes" search="*" externals="no"/>
```

The following snippet updates the items matching the given file name selectors only:

```
<svnupdate path="${documents.dir}">
    <include name="README*"/>
    <include name="RELEASE*"/>
</svnupdate>
```

See Also

- The <svncommit> task lets you commit local modifications to the repository.
- The <svnrevert> task lets you undo local working copy modifications.

SvnAddTask <svnadd>

Class: com.idaremedia.svn4ant.client.solo.SvnAddTask **Category:** Client

Description

The SvnAddTask task (defined `<svnadd>`) lets you schedule new local files to be added to the repository on your next commit. You can optionally commit all modifications from the working copy immediately after the add operation by setting the `commit` parameter to "`yes`". (Note that if you select explicit items for addition, only those items are commited; any other existing modifications are not.)

The `<svnadd>` task has a builtin fileset filter that it will apply to the set of possible files it can add. To filter the items Svn4Ant adds, you can use either a nested `<items>` filter element (the fileset itself), or any of the common `<include>`, `<exclude>` file selectors. If you select a nested file for addition, Svn4Ant will automatically add any parent directories that are currently unversioned.

Usually if a file or directory matches one of the global Subversion ignore patterns, it will not be added. To override this behavior you must set the "`defaultignores`" parameter to "`off`".

Parameters

The following parameters are ***in addition*** to the common parameters inherited by all Svn4Ant client tasks. Read the "Common Client Parameters" section for more information.

Attribute	Description	Required
`path`	Path to a local filesystem object. If the `type` option is undefined and this filesystem object does not exist, Svn4Ant assumes you want to add a blank ***file*** at this location. This path must have one of its parent directories already under Subversion control.	Yes
`type`	Set to "`dir`" or "`file`" to have Svn4Ant automatically create the right type item named by '`path`'. Ignored if the item already exists. Note that files are left empty (0K).	No; defaults "`file`"
`commit`	Set to "`yes`" to have Svn4Ant automatically commit the added items. If you have not selected specific items, all modifications in/under path are commited; otherwise, only the newly added items are commited.	No; defaults "`no`"
`defaultignores`	Set to "`no`" to add items normally ignored by Subversion clients (as controlled by your ***local*** Subversion config or the '`configdir`' parameter).	No; defaults "`yes`"
`recurse`	Set to "`no`" to block a recursive additions. If recursion is blocked, only the named path ***item*** is commited; for directories—no directory contents are added but required parent directories will be.	No; defaults "`yes`"

Attribute	Description	Required
`force`	Set to “`yes`” if you want to automatically add all currently unversioned items in a newly added directory or if you want to automatically add the parents of a new nested directory. You should combine this option with the ‘`recurse`’ parameter when adding a directory tree. Required with a nested `<items>` element (which selects the items to add) to ensure parent directories are added if necessary.	No; defaults “`no`”

Nested Element: `<message>`

The `<message>` element lets you specify a more complex commit message than the shorthand `message` parameter. Ignored unless you immediately commit your additions.

Nested Element: `<items>`

The `<items>` element is the add task’s builtin Ant `<fileset>` that lets you specify filters like patternsets and special selectors to create the set of added files and directories. ***Only items included by the filter will be added.*** The format of the the `<items>` element is exactly that of the standard Ant `<fileset>` except the required `dir` parameter is automatically assigned the value of the parent add task’s ‘`path`’ parameter; read the Ant documentation for further information on file sets.

Nested Element: `<include>`

The `<include>` element lets you quickly define an file name selector for the add task’s builtin fileset. The format is the exactly that of the standard Ant fileset `<include>` element.

Nested Element: `<exclude>`

The `<exclude>` element lets you quickly define an exclusion file name selector for the add task’s builtin fileset. The format is the exactly that of the standard Ant fileset `<exclude>` element.

Examples

The following snippet schedules the local file “branch-readme.txt” for addition. No changes to the remote repository are performed so no credentials are specified:

```
<svnadd path="${meta.dir}/branch-readme.txt"/>
```

The following snippet extends the previous example and immediately commits the new file to the “`devbox.repo`” repository:

```
<svnadd credential="devbox.repo"
    path="="${meta.dir}/branch-readme.txt" commit="yes">
    <message>Branch@0: ${branch.label}</message>
</svnadd>
```

The following snippet adds and immediately commits any new report templates defined by the nested `<items>` fileset. Only items matching the pattern “`*.*,tmpl`” that are not already under Subversion control are scheduled for addition:

```
<svnadd credential="reports.repo" path="${work.dir}" commit="yes">
    <message>+ New ${TYPE} templates for ${DATE}</message>
    <items>
      <include name="**/*.*,tmpl"/>
      <exclude name="**/.*/"/>
      <not>
         <isversioneditem/>
      </not>
    </items>
</svnadd>
```

See Also

- The `<svndelete>` task lets you delete an item from a repository.
- The `<svnrevert>` task lets you undo local additions.
- The `<svnimport>` task lets you add a new unversioned directory and its contents at once.

SvnDeleteTask <svndelete>

Class: com.idaremedia.svn4ant.client.solo.SvnDeleteTask **Category:** Client

Description

The SvnDeleteTask task (defined `<svndelete>`) lets you schedule existing items to be removed from the repository on your next commit or lets you immediately remove items from the repository. For local deletes, you can optionally commit all modifications immediately by setting the `commit` parameter to "`yes`".

The `<svndelete>` task has a builtin fileset filter that it will apply to the set of possible files it can delete. To filter the items Svn4Ant deletes, you can use either a nested `<items>` filter element (the fileset itself), or any of the common `<include>`, `<exclude>` file selectors. Note that your fileset selectors must be internally consistent with the deletion of directories; so for example, you cannot select a directory for deletion while excluding an item inside of that directory from deletion (Svn4Ant will delete the item).

For local deletions, you can also ask Svn4Ant to leave the now unversioned items in the local filesystem.

Parameters

The following parameters are ***in addition*** to the common parameters inherited by all Svn4Ant client tasks. Read the "Common Client Parameters" section for more information.

Attribute	Description	Required
`path`	Path to a local filesystem object. This item must exist and be under Subversion control. If one or more nested `<item>`s are defined, this path is used as the base path for all of them.	Yes; unless a nested `<url>` is defined.
`commit`	Set to "`yes`" to have Svn4Ant automatically commit the deletions. If you have not selected specific items, all modifications in/under '`path`' are commited. Ignored for remote deletions (these are always immediate).	No; defaults "`no`"
`removefiles`	Set to "`no`" to keep the now unversioned items from being deleted locally. Ignored for remote deletions.	No; defaults "`yes`"
`recurse`	Only applies to post-deletion commit step. Set to "`no`" to block recursive commits. If recursion is blocked, only the named path ***item*** is commited; for directories—no directory contents are added but required parent directories will be. Ignored for remote deletions.	No; defaults "`yes`"
`force`	Set to "`yes`" if you want to force the operation to run when it would not normally; for example, if a the item to be deleted has local modifications, normally a delete would fail.	No; defaults "`no`"

Nested Element: <message>

The <message> element lets you specify a more complex commit message than the shorthand message parameter. Ignored unless it's a remote deletion or you immediately commit your deletions.

Nested Element: <url>

The <url> element lets you define a remote item for deletion. You can include any number of remote items for deletion (but not in combination with local files). If the URLs are not absolute repository locators, you must specify a base URL through an <svnserver> reference.

You must define the <url> contents as the text between the open and close url elements like:
<url>svn://repos/myproject/thing/to/delete</url>
or as the value of a single "value" attribute like:
<url value="svn://repos/myproject/thing/to/delete"/>.

Nested Element: <items>

The <items> element is the delete task's builtin Ant <fileset> that lets you specify filters like patternsets and special selectors to create the set of deleted files and directories. ***Only items included by the filter will be deleted***. The format of the the <items> element is exactly that of the standard Ant <fileset> except the required dir parameter is automatically assigned the value of the add's path parameter; read the Ant documentation for further information on file sets.

Nested Element: <include>

The <include> element lets you quickly define an file name selector for the delete task's builtin fileset. The format is the exactly that of the standard Ant fileset <include> element.

Nested Element: <exclude>

The <exclude> element lets you quickly define an exclusion file name selector for the delete task's builtin fileset. The format is the exactly that of the standard Ant fileset <exclude> element.

Examples

The following snippet schedules the local file "branch-readme.txt" for deletion. No changes to the remote repository are performed so no credentials are specified:

```
<svndelete path="${meta.dir}/branch-readme.txt"/>
```

The following snippet is a variation of the previous example that deletes the file directly from the "devbox.repo" repository:

```
<svndelete credential="devbox.repo" message="Frozen ${BUILDSTAMP}">
    <url>${$svnurl:devbox.repo}/${meta.loc}/branch-readme.txt</url>
</svndelete>
```

The following snippet purges a set of tagged nightly builds from a remote repository. The list of target items is determined elsewhere (off-screen left):

```
<foreach i="tagrelease" list="${outofdate.snaps}" tryeach="yes" mode="local">
  <svndelete credential="build.repo">
     <message text="## Automatic purge because ${cause}"/>
     <url value="${tagrelease}"/>
  </svndelete>
  <emit msgid="info.vcs.purge" msgarg1="${tagrelease}" msgarg2="${cause}"
      from="news.developers"/>
</foreach>
```

See Also

- The `<svncopy>` task lets you resurrect an item that has been deleted.
- The `<svnrevert>` task lets you undo local additions.

SvnRevertTask <svnrevert>

Class: com.idaremedia.svn4ant.client.solo.SvnRevertTask **Category:** Client

Description

The SvnRevertTask task (defined `<svnrevert>`) lets you undo any local changes to your working copy. The reverted items (if they were under Subversion control) are returned to their BASE values.

Parameters

The following parameters are ***in addition*** to the common parameters inherited by all Svn4Ant client tasks. Read the "Common Client Parameters" section for more information.

Attribute	Description	Required
`path`	Path to a local working copy item (file or directory). This item must exist.	Yes.
`recurse`	Set to "`no`" to block a recursive revert operation. For directories only its direct contents are reverted (along with any modifications on the directory itself).	No; defaults to "`yes`.

Nested Elements

The `<svnrevert>` task does not support any nested elements.

Examples

The following snippet will revert the local changes to some documentation files if anything fails before the final commit can occur. Note that a default `<svnserver>` was defined so no task-specific credential is necessary:

```
<svnserver id="svnrepo" default="yes"…/>
…
<protect …>
    [set of tasks that modify local working copy that can fail…]
    <svncommit path="${docs}" message="…"/>
    <iferror …>
        <svnrevert path="${docs}" haltiferror="no"/>
    </iferror>
</protect>
```

SvnCommitTask <svncommit>

Class: com.idaremedia.svn4ant.client.solo.SvnCommitTask **Category:** Client

Description

The SvnCommitTask task (defined `<svncommit>`) lets you commit your working copy modifications to the shared repository. You can commit all modifications from a working copy or select a file subset using standard Ant file set selectors and patternsets. To tell Svn4Ant to automatically add all unversioned files in the working copy before the actual commit, set the `addlocals` parameter to "`yes`". To ensure new, unversioned parent directories are also automatically added before the commit, combine the `addlocals` parameter with the `force` parameter; set both to "`yes`".

To commit multiple independent working copies that share a common parent directory, you can use the `search` parameter. Set the value of the `search` parameter to the maximum number of subdirectory levels the operation should search for working copy roots. Each root the operation finds it will commit using the parameters you've defined on the `<svncommit>` instance. Set the `search` parameter to "`*`" to search for all roots below the parent directory.

The `<svncommit>` task has a builtin fileset filter that it will apply to the set of possible files it can commit. To filter the items Svn4Ant commits, you can use either a nested `<items>` filter element (the fileset itself), or any of the common `<include>`, `<exclude>` file selectors. Note that for filtered commits, each item that matches the filter is commited independently; in other words, the commit is not a single atomic operation.

Parameters

The following parameters are ***in addition*** to the common parameters inherited by all Svn4Ant client tasks. Read the "Common Client Parameters" section for more information.

Attribute	Description	Required
`path`	Path to a local filesystem object. If the `search` option is disabled, this filesystem object (a file or directory) must be under Subversion control. If the `search` parameter is enabled, this option should be the path to an existing directory.	Yes
`addlocals`	Set to "`yes`" to have Svn4Ant scan '`path`' for all unversioned items and add them before the commit.	No; defaults "`no`"
`search`	Set to the number of levels that Svn4Ant should decend looking for independent working copy roots. If set to "`*`" Svn4Ant will look for all working copy roots below the directory specified in the '`path`' parameter.	No; defaults "`0`"
`keeplocks`	Set to "`yes`" to retain any existing item locks after the commit completes.	No; defaults "`no`"
`recurse`	Set to "`no`" to block a recursive commit. If recursion is blocked, only the named path ***item*** is commited; for directories—no directory contents are commited.	No; defaults "`yes`"

Attribute	Description	Required
`force`	Set to "`yes`" to force this operation to automatically add unversioned parent directories (up to the root) of any items added with the `addlocals` option. Note that if the item resides inside and externally linked root, ***the link's*** repository is affected. If the '`externals`' option is off, forcing has no affect on such items.	No; defaults "`no`"
`externals`	Set to "`no`" to disable update of items defined by the special "`svn:externals`" directory property.	No; defaults "`yes`"

Nested Element: `<message>`

The `<message>` element lets you specify a more complex commit message than the shorthand `message` parameter.

Nested Element: `<items>`

The `<items>` element is the commit task's builtin Ant `<fileset>` that lets you specify filters like patternsets and special selectors to create the set of commited files and directories; ***only items included by the filter will be commited***. The format of the the `<items>` element is exactly that of the standard Ant `<fileset>` except the required `dir` parameter is automatically assigned the value of the commit's `path` parameter; read the Ant documentation for further information on file sets.

Nested Element: `<include>`

The `<include>` element lets you quickly define an file name selector for the commit task's builtin fileset. The format is the exactly that of the standard Ant fileset `<include>` element.

Nested Element: `<exclude>`

The `<exclude>` element lets you quickly define an exclusion file name selector for the commit task's builtin fileset. The format is the exactly that of the standard Ant fileset `<exclude>` element.

Examples

The following snippet commits all recorded modifications of a working copy `${workdir}` to the shared repository. Only edits the working copy knows about (i.e. done using Subversion commands) are processed. The "`devbox.repo`" credential is used to respond to any repository authentication challenge:

```
<svncommit credential="devbox.repo" path="${workdir}" message="…"/>
```

The following snippet commits all the working copies located under the directory defined by the "`${modules}`" property. Svn4Ant uses the default credential (see `<svnserver>`) to respond to any authentication challenge and an empty commit message:

```
<svncommit path="${modules.dir}" search="*"/>
```

The following snippet commits any modifications and newly added items (including directories) to the repository. Directories defined by `svn:externals` properties are not included in commit:

```
<svncommit path="${meta.dir}" addlocals="yes" force="yes" externals="no">
    <message>Nightly: ${build.label}</message>
</svncommit>
```

The following snippet commits the working copy roots located directly inside the `${workspace.dir}` directory. The `<isworkingcopyroot>` is custom file selector included in Svn4Ant that matches any directory that is a working copy root:

```
<svncommit path="${workspace.dir}" message="…">
    <depth max="0"/>
    <isworkingcopyroot externals="yes"/>
</svncommit>
```

See Also

- The `<svn>` task lets you get a status of all the modified files in your working copy.
- The `<svnrevert>` task lets you undo local modifications.

SvnTransferSet <svntransferset>

Class: com.idaremedia.svn4ant.client.transfer.SvnTransferSet **Category:** Client

Description

The SvnTransferSet type (defined `<svntransferset>`) lets you define a set of reusable transfer instructions independent of the Ant scripts that use the instructions. You could, for example, maintain the set of items that make up you final exported product package in a separate configuration file from the Ant script that builds the package.

You can use a `<svntransferset>` with most Svn4Ant operations that support moves or copies of multiple remote items in a single go; for example, both the `<svnbranch>` and `<svnexport>` operations support external transfer sets.

Parameters

Attribute	Description	Required
`from`	Path to a local properties file that contains the set of transfer instructions. Each line in the file should be a space delimited file with at least two entries: the "to" item followed by an optional revision and then the "from" item. This format is very similar to the simple (pre 1.5) `svn:externals` line format.	Yes unless a nested `<item>` is used.

Nested Element: `<item>`

The `<item>` element lets you specify a single transfer instruction. If you do not define all parameters of the nested item, the controlling component will decide what the default values should be. For instance, the `<svnexport>` task will use the basename of the source '`from`' parameter as the name of the local target.

Parameters

Attribute	Description	Required
`from`	Path to repository directory on which the local working copy is based. This repository location must exist.	Yes
`to`	Path to a local directory; Svn4Ant will create this directory if necessary. If this is a relative path, it is created relative to the parent <svncheckout> task's 'to' parameter.	No; defaults to basename of directory.
`revision`	Set to the revision you want the controlling operation to use. This parameter will have the same limitations as the '`revision`' option of the controlling component.	No; defaults "`HEAD`".

Examples

The following snippet loads a set of transfer instructions from an external file "`release.def`" that is maintained independent from the Ant scripts:

```
<svntransferset id="release-packages" file="${buildmeta.dir}/release.def"/>
```

The following snippet defines a set of transfer instructions that creates a single directory comprised of working copy roots from various locations in a single parent repository:

```
<svntransferset id="log4ant.modules">
    <item from="apis/jware/tags/r1.4" to="apis"/>
    <item from="antxtras/core/main" to="core" revision="123"/>
    <item from="antxtras/feedback/main" to="feedback" revision="123"/>
</svntransferset>
```

See Also

- The `<svncheckout>` task lets you checkout a set of related repository directories into a single parent directory.
- The `<svnbranch>` task lets you copy a repository item and move all external links with it.

SvnCopyTask <svncopy>

Class: com.idaremedia.svn4ant.client.solo.SvnCopyTask **Category:** Client

Description

The SvnCopyTask task (defined `<svncopy>`) lets you copy a local working copy item or a remote repository item to another location. Like the source, the destination can be a local working copy directory or a repository directory. You can use a single `<svncopy>` command to copy multiple items to a single local location in one operation. The copies are checked out to a shared parent directory (itself not under version control).

For each remote copy (source and destination are URLs), the locations must belong to the same repository. For now, this is a limitation of Subversion itself, not SVNKit or Svn4Ant.

Note that although `<svncopy>` lets you specify multiple copies together, ***each copy is done independently***. This means that if a single copy operation fails, all preceding successful copies would have been already commited (in the case of remote copies ***and*** automatically commited local copies).

Parameters

The following parameters are ***in addition*** to the common parameters inherited by all Svn4Ant client tasks. Read the "Common Client Parameters" section for more information.

Attribute	Description	Required
`from[path]` or `fromurl`	Source to copy; either a local working copy item or a repository url. This item must exist. If doing a remote copy, the source url must belong to the same repository as the destination.	Yes unless a nested `<item>` is used.
`to[path]` or `tourl`	Path to a destination local directory or remote location. For local directory copies Svn4Ant will create this directory if necessary. If the named item is an existing directory, the new item is copied ***into*** it, so you cannot point to an existing directory that contains an item already named the same as the source.	Yes unless a nested `<item>` is used.
`revision`	Set to the revision you want copied. If copying from a working copy and `revision` is different from the `WORKING` revision, the copied files are pulled from the repository.	No; defaults "`HEAD`" or "`WORKING`"
`commit`	Set to "`yes`" to have Svn4Ant automatically commit a copied working copy area. Ignored for remote copies (these are always immediate).	No; defaults "`no`"
`precommithook`	Set to the name of a local `macrodef` that Svn4Ant should execute before doing a commit of a ***local*** copy. Automatically enables the commit option. Ignored for remote copies. Expects named macrodef to accept a single 'workingcopy' attribute that contains the path to the local copy.	No

Attribute	Description	Required
`force`	Set to "`yes`" to force this operation to run for a working copy to working copy operation. Will overwrite existing target files if turned on.	No; defaults "`no`"

Nested Element: `<message>`

The `<message>` element lets you specify a more complex commit message than the shorthand `message` parameter. Ignored unless it's a remote copy or you immediately commit your additions.

Nested Element: `<item>`

The `<item>` element lets you specify a single copy instruction that occurs as part of a set of different copies. For local copies the targets will all reside under a single parent (unversioned) directory. You would use `<item>` elements to snapshot a set of related projects in a single pass. If you do not define an item's `to` parameter, Svn4Ant will use the base name of the copied directory.

Parameters

Attribute	Description	Required
`from[path]` or `fromurl`	Source to copy; either a local working copy item or a repository URL. This item must exist. If doing a remote copy, the source URL must belong to the same repository as the destination.	Yes
`to[path]` or tourl	Path to a destination local directory or remote location. For local directory copies Svn4Ant will create this directory if necessary. If the named item is an existing directory, the new item is copied ***into*** it, so you cannot point to an existing directory that contains an item already named the same as the source.	Yes
`revision`	Set to the revision you want copied. If copying from a working copy and `revision` is different from the `WORKING` revision, the copied files are pulled from the repository.	No; defaults "`HEAD`" or "`WORKING`"

Examples

The following snippet copies a fixed tag revision to another area of the repository location defined by the "`devbox.repo`" credential:

```
<svncopy credential="devbox.repo" fromurl="tags/${stable}" tourl="demos/${stamp}"/>
```

The following snippet copies a project's trunk subdirectory at the specific repository revision `123`.

```
<svncopy username="anonymous" password="anonymous"
    from="${repo}/proj/trunk" to="${tmpdir}/proj-123" revision="123"/>
```

See Also

- The `<svnmove>` task lets you move a repository item to another location with repository.
- The `<svnbranch>` task lets you copy a repository item and move all links with it.
- The `<svnexport>` task lets you export part of a repository for use outside Subversion.

Description

The SvnBranchTask task (defined `<svnbranch>`) lets you copy a remote repository item to another repository location and update internal and external links in one step. Unlike the SvnCopyTask (`<svncopy>`), the SvnBranchTask will move internal link references (relative to the URL being copied) to the new location, and will optionally pin external references (as is often required for tagging). The base model for the `<svnbranch>` task is the '`svncopy.pl`' script that comes with the standard Subversion client distribution (see the `contrib` directory).

The `<svnbranch>` task tries to create the entire branch, including fixed externals, in a single atomic step. But if you specify a target URL whose intermediate parent directories do not exist in the repository, `<svnbranch>` will create the new remote parent directories before attempting the branch (using the equivalent of an '`svn mkdir`' operation). If the branch step fails after the parent directories have been created, `<svnbranch>` will not try to remove them.

In order to do a single commit, `<svnbranch>` attempts to checkout, non-recursively, the parent of the branch and do a remote to local copy (thereby creating a single add operation to the parent with all externals and nested modifications attached). If your branch's parent directory contains very large numbers of ***files*** (directories are not checked-out so they don't matter), you might have to find another mechanism for branching or be prepared to wait for the checkout of the parent. This method works well for repository setups where "tags" and "branches" each reside under a single parent directory.

Although you can create a new branched area from multiple sources, each source must belong to the same repository; this restriction does not apply to external references.

Parameters

The following parameters are ***in addition*** to the common parameters inherited by all Svn4Ant client tasks. Read the "Common Client Parameters" section for more information.

Attribute	Description	Required
`from`	Path to source in remote location to copy. This item must exist and belong to the same repository as the destination.	Yes unless a nested `<item>` is used.
`to`	Path to a destination remote location.	Yes
`revision`	Set to the revision you want branched. If `revision` is different from the HEAD revision, any pinned externals are pinned relative to this revision (or more accurately pinned to the last revision just before your stated revision's timestamp).	No; defaults "`HEAD`"

Attribute	Description	Required
`tempdir`	Points to a local directory that Svn4Ant should use to create its working copy for the branch. Svn4Ant will create this directory if necessary. If defined, Svn4Ant will not delete this directory automatically (even if it had to create the directory itself).	No; defaults to a temporary directory
`clean`	Set to "`yes`" to have Svn4Ant purge the contents of an existing scratch directory before attempting its checkouts.	No; defaults to "`no`"
`externals`	Set to either "`keep`" or "`pin`" to tell Svn4Ant what to do with unpinned external links (svn:externals where the URL does not refer back to the area being branched). Note that this setting has no impact on internal links – which are always moved and not pinned.	No; defaults to "`pin`"
`precommithook`	Set to the name of a local `macrodef` that Svn4Ant should execute before doing a commit of the new branch. Executed ***even if*** dryrun is enabled. Expects named macrodef to accept a single '`workingcopy`' attribute that contains the path to the local new branch directory.	No
`dryrun`	Set to "`yes`" to block the final commit of the branch working copy. This lets you review or do additional manual tweaks to the branch (or tag) before checking it in for the first time.	No; defaults "`no`"

Nested Element: `<message>`

The `<message>` element lets you specify a more complex commit message than the shorthand `message` parameter. Ignored if the '`dryrun`' parameter is turned on.

Nested Element: `<item>`

The `<item>` element lets you specify a single copy instruction that occurs as part of a set of different copies. You would use `<item>` elements to branch a set of related projects in a single pass. If you do not define an item's `to` parameter, Svn4Ant will use the base name of the copied path. All externals are handled relative to their owning item not the parent `<svnbranch>`; this lets you combine completely different revision histories into a single branch.

Parameters

Attribute	Description	Required
`from`	Remote source to copy. This item must exist and belong to the same repository as the destination.	Yes
`to`	Relative path within branch area (relative to the main 'to' parameter of the `<svnbranch>` itself). This can be the empty string in which case the contents are copied directly into the branch root directory.	No

Attribute	Description	Required
`revision`	Set to the revision you want copied. If not defined, inherits the revision of the main `<svnbranch>` itself. Note that all externals within this part of the branch will be pinned relative to this revision (actually this revision's timestamp).	No; defaults to branch revision

Nested Element: `<items>`

The `<items>` element lets you reuse a set of copy instructions defined elsewhere as an `<svntransferset>`. Because you can load an `<svntransferset>` from a file or other external source, the `<items>` element lets you alter the list of sources to branch independent of the main script file. See the description of the `<svntransferset>` item for more information.

Examples

The following snippet creates a branch of a trunk location defined by the "`my.repos`" credential (an `<svnserverset>`). Both the trunk's source and all externals references are left as-is; internals references are moved to refer to the branch's location:

```
<svnbranch credential="my.repos" externals="keep"
   from="myproject/trunk" to="myproject/branches/${branch.name}"
   message="Workstream for myproject ${branch.name}"/>
```

The following snippet creates a snapshot of a trunk location defined by the "`my.repos`" credential. The trunk's source is left as-is; all externals references are pinned to their revision as of the time of the last change to the trunk's HEAD:

```
<svnbranch credential="my.repos" externals="pin"
   from="myproject/trunk" to="myproject/snapshots/${snapshot.name}"
   message="Snapshot for myproject (${snapshot.name})"/>
```

The following snippet combines a call to `<svnbranch>` with one to `<svnretire>` to create a offspring branch from a newly frozen branch. Basically the current branch is baselined, and a new branch created from that baseline. Note that the new branch's externals start in a pinned form; they must be explicitly unpinned if a change to the branch requires it (thus linking the "unpinning" to the change that required it).

```
<svnretire credential="my.repos"
  from="${product}/branches/${branch.name}"
  to="${product}/tags/${branch.name}"
  message="Retired ${product.longname} ${branch.name}"
  />
<svnbranch credential="my.repos"
    from="${product}/tags/${branch.name}"
    to="${product}/branches/${newbranch.name}"
    message="Emergency fixes for ${branch.name}"/>
```

The following snippet creates a branch from a collection of sources. All external references are pinned relative to their owning item's revision and their source's repository history.

```
<svnbranch credentials="antx.repos" to="branches/AntX_0.5.${version.ebr}">
    <item from="antx-builder/trunk" revision="1252" to=""/>
    <item from="antx-core/trunk" revision="1234" to="modules/core"/>
    <item from="antx-fixture/trunk" revision="1236" to="modules/fixture"/>
    <item from="antx-flowcontrol/trunk" revision="1245" to="modules/flowcontrol"/>
    <item from="antx-install/trunk" revision="1246" to="modules/install"/>
    <message>Branch for emergency fixes of AntX_0.5.x</message>
    </svnbranch>
```

See Also

- The `<svncopy>` task lets you copy a repository item to another location within the repository without touching externals.
- The `<svnretire>` task lets you pin all externals references then move a repository item to another location (useful for tagging and/or baselining).
- The `<svnexport>` task lets you export part of a repository for use outside Subversion.

SvnMoveTask <svnmove>

Class: com.idaremedia.svn4ant.client.solo.SvnMoveTask **Category:** Client

Description

The SvnMoveTask task (defined `<svnmove>`) lets you move a repository item (file or directory) to a new repository location; only server-side moves are supported currently. The SvnMoveTask does not attempt to move internal 'svn:externals' (links to directories under the same root directory) so it is only appropriate for repository moves that require no such complexity.

Parameters

The following parameters are ***in addition*** to the common parameters inherited by all Svn4Ant client tasks. Read the "Common Client Parameters" section for more information.

Attribute	Description	Required
`fromurl`	Set to the source repository location. If this is a relative path it is resolved with respect to the URL defined by your '`credential`'.	Yes
`tourl`	Set to the destination repository location. All intermediate parent directories must exist in the repository.	Yes
`revision`	Set to a specific source revision to move.	No; defaults to "`HEAD`"

Nested Element: <message>

The `<message>` element lets you specify a more complex commit message than the shorthand `message` parameter.

Examples

The following snippet migrates a particular version of a library from a "latest" state to its final stable state by renaming the directory the library is stored under:

```
<svnserver id="my.repo" isdefault="true"…/>
<svncat>
    <item from="libs/${lib}/latest/VERSION" toproperty="_lib.version"/>
</svncat>
<svnmove fromurl="libs/${lib}/latest" tourl="libs/${lib}/${_lib.version}"
   message="Library ${lib} baselined to version ${_lib.version}"/>
```

See Also

- The `<svnretire>` task lets you move a repository tree to another location within repository including any internal (self-referential) links.

SvnRetireTask <svnretire>

Class: com.idaremedia.svn4ant.client.branch.SvnRetireTask **Category:** Client

Description

The SvnRetireTask task (defined `<svnretire>`) lets you move a repository directory to a new, but final, repository location. All internal (self-referential) links are updated to the new location and all external links are automatically pinned. Unlike the simple SvnMoveTask (`<svnmove>`), SvnRetireTask is meant for moving server-side directory trees only; you should move single files with the regular move command.

You would use the `<svnretire>` task to tag or baseline a changing branch in your repository (in this case the typical "trunk" is also considered a branch). The retire process combines three steps: the initial internal links repointing step, the external links pinning step, and the subsequent server-side move operation. Note that the pinning and link resets come ***before the final move*** in their own committed step (think of it as "the retirement prep step"). If the subsequent move fails for some reason (but the prep step succeeded), you only have to fix the cause of the move problem and then do a ***simple*** `` `svn move` `` operation of the branch to its final tagged location (the internals have already been preset by Svn4Ant).

The `<svnretire>` task expects the directory it is operating on to be frozen during its execution. In other words, no new unknown commits will occur against the directory or its contents from the time the task begins to the time it finishes (otherwise you'll end up trying to move something that does not represent the last modification to that directory tree). By naming this task `<svnretire>` we want to convey this "final-ness" of changes to that directory tree.

Parameters

The following parameters are ***in addition*** to the common parameters inherited by all Svn4Ant client tasks. Read the "Common Client Parameters" section for more information.

Attribute	Description	Required
`from`	Path to source in remote location to move. This item must exist and belong to the same repository as the destination.	Yes
`to`	Path to a destination remote location.	Yes
`revision`	Set to the revision you want retired. If `revision` is different from the `HEAD` revision, any pinned externals are pinned relative to this revision (or more accurately pinned to the last revision just before your stated revision's timestamp).	No; defaults "`HEAD`"
`prepmessage`	Set to a unique commit message for any pre-move changes that the command must perform (like pinning). If you define the general '`message`' parameter but not this one, Svn4Ant will use a slight variant of '`message`' for a prep-step message.	No

Attribute	Description	Required
`tempdir`	Points to a local directory that Svn4Ant should use to create its working copy for the pre-move tweaking. Svn4Ant will create this directory if necessary. If defined, Svn4Ant will not delete this directory automatically (even if it had to create the directory itself).	No; defaults to a temporary directory
`clean`	Set to "`yes`" to have Svn4Ant purge the contents of an existing scratch directory before attempting its checkouts.	No; defaults to "`no`"
`dryrun`	Set to "`yes`" to do a test run. Svn4Ant will checkout the existing source and create a retire report containing what links would have to move and what links would be pinned in the real operation.	No; defaults "`no`"

Nested Element: `<message>`

The `<message>` element lets you specify a more complex commit message than the shorthand `message` parameter. This is the retirement's final move operation's message. Use the <prepmessage> element for a unique message for any preparatory step commits (like for external link pinning or internal link redirection).

Nested Element: `<prepmessage>`

The `<prepmessage>` element lets you specify a more complex commit message for the prep-step of the retirement operation. Only used if prep is needed.

Examples

The following snippet retires a Proof-of-Concept (PoC) branch to a permanent snapshot:

```
<svnretire credential="my.repos"
     from="${product}/lab/${poc.id}"
     to="${product}/tags/${poc.id}-EOL"
     message="Retired ${poc.id} permanently"/>
```

The following snippet combines a call to `<svnretire>` with one to `<svnbranch>` to create a offspring branch from a newly retired branch. Basically the current branch is baselined, and a new branch created from that baseline.

```
<svnretire credential="my.repos"
  from="${product}/branches/${branch.name}"
  to="${product}/tags/${branch.name}"
  message="Retired ${product.longname} ${branch.name}"
  />
<svnbranch credential="my.repos"
   from="${product}/tags/${branch.name}"
   to="${product}/branches/${newbranch.name}"
   message="Emergency fixes for ${branch.name}"/>
```

See Also

- The `<svnbranch>` task lets you copy a repository tree to a new tree within repository including any internal (self-referential) links.

SvnCatTask <svncat>

Class: com.idaremedia.svn4ant.client.solo.SvnCatTask **Category:** Client

Description

The SvnCatTask task (defined `<svncat>`) lets you copy the contents of one or more remote or working copy files to unversioned files or Ant fixture elements like properties or references. The `<svncat>` task is useful for extracting the contents of disparate versioned files with post-processing from Ant filters without first having to create a scratch working copy just to get at the file. If you do not specify a target file, Svn4Ant will concatenate the file's contents to the standard Ant log console.

Parameters

The following parameters are ***in addition*** to the common parameters inherited by all Svn4Ant client tasks. Read the "Common Client Parameters" section for more information.

Attribute	Description	Required
`from`	Source to concatenate; either a local working copy item or a repository URL. In both cases, this item must exist.	Yes unless nested `<item>`.
`workingcopy`	Set to the local working copy to use as a base for any nested `<item>` elements. Do not use as a substitute for a local source file; use the '`from`' parameter instead.	No
`tofile`	Set to the path to a local file where contents will be stored. Svn4Ant will create this file if it does not exist. Shortcut for a single nested `<item>`.	No; defaults to writing to Ant's console.
`toproperty`	Set to the name of a local property where contents will be stored. This property should not exist in current project. Shortcut for a single nested `<item>`.	
`tovar[iable]`	Set to the name of an iteration variable where contents will be stored. Any existing variable value is replaced unless the '`append`' option is turned on. Shortcut for a single nested `<item>` with a '`tovariable`' parameter.	
`toreference`	Set to the name of a reference where contents will be stored as a String object. An existing reference value is replaced. Shortcut for a single nested `<item>` with a '`toreference`' parameter.	
`append`	Set to "`yes`" to have the source content appended to any existing content in the destination (like a file's current contents).	No; defaults "`no`"
`revision`	Set to the revision you want concatenated. If pulling from a working copy and `revision` is different from the `WORKING` revision, the copied files are pulled from the repository. If there are nested `<item>`s, this revision will be used if a specific revision is not defined for the item.	No; defaults "`HEAD`" or "`WORKING`"

Attribute	Description	Required
`keywords`	Set to "`no`" to disable automatic keyword expansion when applicable.	No; defaults "`yes`".

Nested Element: `<item>`

The `<item>` element lets you specify a single copy instruction that occurs as part of concatenation. In particular, you can specify the source file, the destination (file, property, console, etc.), and any special processing instructions like filter chains. Because each `<item>` is a complete self-contained transfer instruction, a single `<svncat>` can contain multiple nested `<item>`s; this will be more efficient for transfering several remote files as it creates a single connection to the repository and reuses it for each transfer instruction.

Parameters

Attribute	Description	Required
`from`	Source to concatenate; either a local working copy item or a repository URL. This item must exist.	Yes
`tofile`	Path to a local file where contents will be stored. Svn4Ant will create this file if it does not exist.	No; defaults to writing to Ant console.
`toproperty`	Name of local property where contents will be stored. This property should not exist in current project.	
`tovar[iable]`	Name of iteration variable where contents will be stored. Any existing variable value is replaced unless the '`append`' option is turned on.	
`toreference`	Name of local reference where contents will be stored. Any existing reference value is replaced unless the '`append`' option is turned on.	
`append`	Set to "`yes`" to have the source content's appended to any existing content in the destination (like a file's current contents).	No; defaults "`no`"
`revision`	Set to the revision you want concatenated. If pulling from a working copy and `revision` is different from the `WORKING` revision, the copied files are pulled from the repository.	No; defaults "`HEAD`" or "`WORKING`"

Nested Element: `<filterchain>`

The `<filterchain>` element is the standard Ant stream filter component that lets you define a set of filters that are applied to the copied information before being stored in the destination item. (This requires Svn4Ant to save the information in a temporary scratch file which it deletes automatically.) The format is the exactly that of the standard Ant filter chain.

Examples

The following snippet checks out a working copy of the trunk subdirectory in the repository location defined by the “`repo`” credential:

```
<svncat credential="repo" …>
    <item from="docs/README" tofile="${docs}/README">
        <filterchain>
            <expandproperties/>
            <replacetokens begintoken="#" endtoken="#">
                <token key="DATE" value="${TODAY}"/>
            </replacetokens>
        </filterchain>
    </item>
    …
</svncat>
```

The following snippet shows a target that creates a configuration package for each environment named in set of different environment. The script combines a common file using `<svncat>` with additional environment-specific files. This snippet demonstrates how `<svncat>` is often combined with `<svnexport>` to create packages from Subversion based information:

```
<target name"config-packages"
  description="Builds config zip files for each targetted environment">
    <fixturecheck isset="env.dirs" msg="'env.dirs' defined"/>
    <mkdir dir="${conf.d}/common"/>
    <newfile path="${conf.d}/common/version.txt">
      <line value="timestamp=${ISTAMP}"/>
      <line value="svnurl=${$svnurl:my.repo}"/>
      <line value="svnrev=${revision}"/>
    </newfile>
    <svncat credential="my.repo" revision="${revision}">
      <item from="config/shared-config.properties"
        tofile="${conf.d}/common/shared.properties"/>
    </svncat>
    <foreach i="env" list="env.dirs" tryeach="yes" mode="local">
      <echo message="### BUILDING PACKAGE: ${env}" level="info"/>
      <isolate>
        <pathproperty name="env.d" value="${conf.d}/${env}"/>
        <property name="label.config-package"
           value="From:${$svnurl:my.repo}/config/${env}@${revision}"/>
        <mkdir dir="${env.d}"/>
        <copy todir="${env.d}">
          <fileset dir="${conf.d}/common"/>
        </copy>
        <svnexport credential="my.repo" from="config/${env}"
            to="${env.d}" externals="no"/>
        <zip destfile="${conf.d}/${env}.zip" dir="${env.d}"
            comment="${label.config-package}"/>
      </isolate>
    </foreach>
    <echo message="Config Packages: ${conf.d}" level="info"/>
</target>
```

See Also

- The `<svn>` task lets list the contents of a remote repository directory to the Ant console.
- The `<svnexport>` task lets you export part of a repository for use outside Subversion.
- The `<svnprop>` task lets you extract meta property information from Subversion items.

SvnExportTask <svnexport>

Class: com.idaremedia.svn4ant.client.solo.SvnExportTask **Category:** Client

Description

The SvnExportTask task (defined `<svnexport>`) lets you extract parts of a remote repository or a local working copy into a local directory structure or a compressed archive minus all the Subversion administrative files. The extracted files are not part of a working copy – all Subversion administration files and directories are stripped out. For working copy exports, you also have the option of copying files from the source that are not under version control (for example, locally generated output files of an application build process). As a convenience, you can tell Svn4Ant to automatically create a compressed archive of the exported items (the exported directory structure is automatically deleted as Svn4Ant assumes the archive is the desired output).

In addition to single source exports, `<svnexport>` lets you use a nested `<svntransferset>` to export multiple remote sources and/or multiple local working copies to a single target. This is useful for creating a single release tree comprised of independently maintained Subversion projects and locally generated components. You can configure each nested item independently (the surrounding `<svnexport>`'s attributes act as defaults). Note that all sources must reside within the same top-level repository!

Parameters

The following parameters are ***in addition*** to the common parameters inherited by all Svn4Ant client tasks. Read the "Common Client Parameters" section for more information.

Attribute	Description	Required
`from`	Source to export; either a local working copy item or a repository url. These are exclusive options; you can define one or the other (or neither) .	Yes unless a nested `<item>` is used.
`to`	Path to a local directory; Svn4Ant will create this directory if necessary. You should not point to an existing directory unless you want to include the existing files in the exported tree. Alternately, you can use the '`clean`' option to remove the local files. If you define nested `<item>` elements, Svn4Ant uses this parameter as the parent directory into which all items are exported.	Yes unless a ***single*** nested `<item>` is used.
`revision`	Set to the revision you want exported. If you are exporting multiple items, this revision is used as the default for all of them. ***For local exports if this is set to anything other than "`WORKING`" a remote export is done to ensure the right snapshot is exported.***	No; defaults "`HEAD`" or "`WORKING`"
`clean`	Set to "`yes`" to have the export erase ***all*** the contents of an existing local directory before the export operation.	No; default "`no`"

Attribute	Description	Required
addlocals	Set to “yes” to have Svn4Ant scan from for all unversioned items and add them to the exported tree. You can override this setting on each nested item separately. Only used if from is a local directory.	No; defaults “no”
recurse	Set to “no” to block a recursive export. If recursion is blocked, only the ***files*** within the named repository directory are exported (no subdirectories are exported).	No; defaults “yes”
force	Set to “no” to prevent the export from overwriting existing local files with files of the same names from the source.	No; defaults “yes”
externals	Set to “yes” to enable export of items defined by the special “svn:externals” directory property.	No; defaults “no”
eol	Set to the preferred EOL character type for the exported text files. Use if you are creating exported text files for a platform other than the current one. Valid values are: "CRLF", "LF", "CR", and "native".	No; defaults to JVM’s current default

Nested Element: <item>

The <item> element lets you specify a single export instruction that occurs as part of a set of different exports under a single parent directory. You would use <item> elements to export a set of related projects in a single pass. If you do not define an item’s to parameter, Svn4Ant will use the base name of the exported directory (local or remote).

Parameters

Attribute	Description	Required
from	URL to repository item or path to a local item for export. All source repository items or local directories must exist. Svn4Ant will first check to see if there is a local file or directory by this name; if nothing is found, it assumes the value is a remote repository location.	Yes
to	Path to a local directory; Svn4Ant will create this directory if necessary. If this is a relative path, it is created relative to the parent <svnexport> task’s to parameter.	No; defaults to basename of from
revision	Set to the revision you want exported. This parameter has the same limitations as the revision option of the parent task.	No; defaults “HEAD” or “WORKING”
addlocals	Set to “yes” to have Svn4Ant scan from for all unversioned items and add them to the exported tree. You can override this setting on each nested item separately. Only used if from is a local directory.	No; defaults “no”

Nested Element: <items>

The `<items>` element lets you reuse a set of export instructions defined elsewhere as an `<svntransferset>`. Because you can load an `<svntransferset>` from a file or other external source, the `<items>` element lets you alter the list of sources to export independent of the main script file. See the description of the `<svntransferset>` item for more information.

Examples

The following snippet exports a tagged version of a project "myproject" defined by the "`devbox.repo`" credential:

```
<svnexport credential="devbox.repo"
    from="myproject/tags/${stable}" to="${dist.dir}/files/${BUILDSTAMP}"/>
```

The following snippet exports a local directory that contains both versioned files and local additions. Any external references (`svn:external`) are also exported in full. Because the target name ends with the special "+tar+bzip" qualifier, Svn4Ant will automatically tar and bzip the result into an archive:

```
<svnexport from="${website.dir}" to="${ftp.dir}/website-${BUILDSTAMP}+tar+bzip"
    eol="CRLF" addlocals="yes" externals="yes"/>
```

The following snippet exports a single archive composed of bits of three different repositories and some local working copies including local unversioned files:

```
<svnexport credential="login.repo" to="${artifacts.d}/${build.id}-cd+tar+gzip">
   <item url="${src.repos}/svn4ant/component/tags/${rev}" to="svn4ant"/>
   <item url="${www.repos}/wiki/trunk" to="webroot"/>
   <item from="${reports.d}" addlocals="yes"/>
   <item from="${apis.d}" addlocals="yes"/>
</svnexport>
```

See Also

- The `<svnimport>` task lets you import a local directory tree into the repository.
- The `<svncat>` task lets you export ***the contents*** of a single repository file to a local file.
- The `<svnprop>` task lets you extract meta property information from Subversion items.

SvnGetRevisionTask <svnrevget>

Class: com.idaremedia.svn4ant.client.solo.SvnGetRevisionTask **Category:** Client

Description

The SvnGetRevisionTask task (defined `<svnrevget>`) lets you retrieve the revision number associated with a remote repository, a remote repository item, or a local working copy item. This task is useful for capturing revision information for reports, feedback, or to ensure that a long-running script affects a particular revision of the repository. If you do not supply a target fixture element like a property, this task will output the revision number to the Ant console.

Parameters

The following parameters are ***in addition*** to the common parameters inherited by all Svn4Ant client tasks. Read the “Common Client Parameters” section for more information.

Attribute	Description	Required
`path`	Path to a local working copy file or root.	Yes; one or other must be defined (note that ‘`url`’ can be defined via the ‘`credential`’)
`url`	URL to a remote repository or repository item.	
`revision`	The revision identifier. Any symbolic repository revision description is accepted; for example, “`WORKING`” or “`{2005-11-28}`”.	No; defaults to “`HEAD`” for remote, “`WORKING`” for working copy.
`property`	Name of a property to create with revision number. Synonym for the inherited “`revisionproperty`” parameter.	No
`var` or `variable`	Name of a variable property to create with revision number.	No
`reference`	Name of a project reference to create with string containing revision number.	No

Nested Elements

The `<svnrevget>` task does not support any nested elements.

Examples

The following snippet displays the current `HEAD` revision number of the repository at “`svn://assets/graphics`” to the Ant console:

```
<svnrevget url="svn://assets/graphics" credential="repo.login"/>
```

The following snippet captures the most recent revision number associated with a local project’s repository as of yesterday into a project property :

```
<tstamp/>
<svnrevget path="${project.dir}" revision="${DSTAMP}" revisionproperty="OLD_REV"/>
```

SvnDirnameTask <svndirname>

Class: com.idaremedia.svn4ant.client.jsvn.SvnDirnameTask **Category:** Client

Description

The SvnDirnameTask task (defined `<svndirname>`) lets you determine the name that SVNKit is using as the name of a working copy's Subversion admin area. Because it is possible for different Subversion installations to change the name from the standard "`.svn`" to something else (e.g. "`_svn`"), you can use this task to make sure your scripts work in user environments where this might be done. If you do not supply a target fixture element like a property to hold the name, this task will output the name to the Ant console.

Parameters

The following parameters are ***in addition*** to the common parameters inherited by all Svn4Ant client tasks. Read the "Common Client Parameters" section for more information.

Attribute	Description	Required
`property`	Name of a property to create with revision number. Synonym for the inherited "`revisionproperty`" parameter.	No
`var` or `variable`	Name of a variable property to create with revision number.	No
`reference`	Name of a project reference to create with string containing revision number.	No

Nested Elements

The `<svndirname>` task does not support any nested elements.

Examples

The following snippet displays the name of the directory SVNKit assumes contains a working copy's administrative files to the Ant console:

```
<svndirname/>
```

The following snippet captures directory's name to a local project property; then uses that property to setup Ant's default excludes to include that directory explicitly:

```
<svndirname property=".localsvn"/>
<defaultexcludes add="**/${.localsvn}"/>
<defaultexcludes add="**/${.localsvn}/**"/>
```

SvnPropTask <svnprop>

Class: com.idaremedia.svn4ant.client.solo.SvnPropTask **Category:** Client

Description

The SvnPropTask task (defined `<svnprop>`) lets you perform the full set of property management operations on repository and working copy items. The full set of property operations (called "actions") are available from this single task but as a convenience the Svn4Ant client antlib also defines a full set of presetdefs for common operations (like `<svnpropget>` for `action="get"` and `<svnpropset>` for `action="set"`). In all cases, you must provide the specific parameters to the operation using the nested `<property>` element.

For reading operations (like "rev`get`" and "`list`"), the retrieved data is sent to the standard Ant console stream unless you explicitly define one of the supported 'to' parameters. For instance, you can save the last repository log message directly to a local file ('tofile'), to a project property ('toproperty'), or to a mutable AntX property ('tovariable').

The SvnPropTask supports the standard '`--revprop`' option by using unique `action` names for those variants of the regular property operations. So, for example, there is a "`revget`" action to match the regular "`get`" and a "`revlist`" action that matches the regular "`list`" action. Both of the `rev`-variants return repository properties instead of item properties. See the Examples section for some uses for these actions.

Parameters

The following parameters are ***in addition*** to the common parameters inherited by all Svn4Ant client tasks. Read the "Common Client Parameters" section for more information.

Attribute	Description	Required
`action`	Set to the name of the property management operation to perform. Each operation will interpret the nested `<property>` item differently (assuming the operation accepts properties). See the "Action:" sub-sections for the rules governing each operation.	Yes
`path`	Set to the local working copy location against which the operation should be applied.	Yes; one of these is required
`url`	Set to the repository location against which the operation should be applied.	
`tofile`	Set to a file where the command's output is saved.	No
`toproperty`	Set to a property where the command's output is saved.	No
`tovar[iable]`	Set to an AntX variable where the command's output is saved.	No
`toreference`	Set to a reference where the command's output is saved.	No
`append`	Set to "`yes`" to append the command's result to the target fixture or file element's current value. Ignored unless one of the 'to' parameters is defined.	No; defaults "`no`"

Attribute	Description	Required
`revision`	Set to the revision you want targeted for reading. For edit operations on versioned properties (like ‘`propdel`’) this is ignored and for all unversioned actions, this parameter is ignored.	No; defaults “`HEAD`” or “`WORKING`”
`recurse`	Set to “`yes`” to force a recursive property update or deletion. For listing operations, setting this to “yes” will concatenate all the results from all the elements.	No; defaults “`no`”
`force`	Set to “`yes`” to force this operation to execute.	No; defaults “`no`”
`strict`	Set to “`yes`” to have the task display just the Subversion query results (no decorating titles or messages).	No; default “`no`”

Action: “`action=get`” [presetdefs: <svnpropget>, <svnpget>]

The “`get`” action lets you retrieve the value of one or more versioned properties to your Ant environment. You can define any number of properties for retrieval; the values will be concatenated to the target fixture item (property, variable, file, etc.) or to the Ant console. If you’ve defined the ‘`revision`’ parameter, Svn4Ant will retrieve that specific revision’s property value. Use the parent task’s ‘`strict`’ parameter to remove any decorative message text from the returned values.

Nested Element: <`property`> [1..*]

Attribute	Description	Required
`name`	Set to the name of the versioned property to get.	Yes

Action: “`action=set`” [presetdefs: <svnpropset>, <svnpset>]

The “`set`” action lets you update an existing or create a new versioned property on a local working copy item or directly on a repository item. You can define any number of nested properties although for remote sets the properties are set one at a time so the entire operation is ***not*** atomic.

Nested Element: <`property`> [1..*]

Attribute	Description	Required
`name`	Set to the name of the versioned property to update or create.	Yes
`value`	Set to the new value of the property. Note you can use the Svn4Ant `${$loadfile:}` value URI to set the property to the contents of a local file.	No; defaults to the empty string.

Action: "`action=list`" [presetdefs: <svnproplist>, <svnplst>]

The "`list`" action lets you retrieve the value of each of an item's versioned properties to your Ant environment. If you've defined the '`revision`' parameter, Svn4Ant will retrieve that specific revision's property list. This action does not use the `<property>` element. Use the parent task's '`strict`' parameter to force the `<svnprop>` task to return the values in exactly the form returned by the repository.

Action: "`action=del`" [presetdefs: <svnpropdel>, <svnpdel>]

The "`del`" action lets you delete one or more versioned properties from an item either in a local working copy or in a repository. You can define any number of nested properties for deletion although for remote deletes the properties are removed one at a time so the entire operation is ***not*** atomic. If the named property does not exist on the item, the operation has no effect on the working copy or repository.

Nested Element: `<property>` [1..*]

Attribute	Description	Required
`name`	Set to the name of the versioned property to delete.	Yes

Action: "`action=revget`" [presetdefs: <svnrevpropget>, <svnrpget>]

The "rev`get`" action lets you retrieve the value of one or more unversioned repository properties to your Ant environment. You can define any number of properties for retrieval; the values will be concatenated to the target fixture item (property, variable, file, etc.) or to the Ant console. Use the parent task's '`strict`' parameter to remove any decorative message text from the returned values.

Nested Element: `<property>` [1..*]

Attribute	Description	Required
`name`	Set to the name of the unversioned repository property to get.	Yes

Action: "`action=revset`" [presetdefs: <svnrevpropset>, <svnrpset>]

The "rev`set`" action lets you update an existing or create a new unversioned repository property. You can define any number of properties to be set although the task will set one repository property at a time so the entire operation is ***not*** atomic.

Nested Element: `<property>` [1..*]

Attribute	Description	Required
`name`	Set to the name of the unversioned repository property to update or create.	Yes

Attribute	Description	Required
`value`	New value of the property. Note you can use the Svn4Ant `${$loadfile:}` value URI to set the property to the contents of a local file.	No; defaults to empty string.

Action: “`action=revdel`” [presetdefs: none]

The “revdel” action lets you delete one or more unversioned repository properties. You can define any number of nested properties for deletion although the properties are removed one at a time so the entire operation is ***not*** atomic.

Nested Element: `<property>` [1..*]

Attribute	Description	Required
`name`	Set to the name of the unversioned repository property to delete.	Yes

Action: “`action=revlist`” [presetdefs: <svnrevproplist>, <svnrplst>]

The “revlist” action lets you retrieve the valuse of each of repository’s unversioned properties to your Ant environment. This action does not use the `<property>` element. Use the parent task’s ‘`strict`’ parameter to remove any decorative message text from the returned values.

Examples

The following snippet dumps the values of two unversioned properties, `svn:log` and `svn:author` to the Ant console:

```
<svnprop credential="login.anon" action="revget" strict="yes">
    <property name="svn:log"/>
    <property name="svn:author"/>
</svnprop>
```

The following snippet is similar to the previous snippet except it lists all of the current unversioned properties to a single file. The snippet uses one of the Svn4Ant presetdefs for the the svnprop task, `svnrevproplist`.

```
<svnrevproplist credential="login.anon" action="revget" strict="yes">
    url="http://svn.lab:9090/repos/javalibraries"
    username="public" password="public"
    strict="yes"
    tofile="${build.d}/revprops.txt"
    feedback="verbose"/>
```

The following snippet updates the HEAD version two versioned properties, ‘`prj:copyright`’ and ‘`prj:license`’ associated with the trunk directory of a project ‘`myproj`’. The snippet uses one of the Svn4Ant presetdefs for the svnprop task, `svnpropset`:

```
<svnpropset credential="my.repo" url="myproj/trunk">
    <property name="prj:copyright" value="(c) Copyright 2008 Me, myself, and I"/>
    <property name="prj:license" value="FSF_GNU_3"/>
</svnpropset>
```

See Also

- The `<svncommit>` task lets you commit local property modifications to the repository.
- The `<svnrevert>` task lets you undo any local property modifications.

Svn4Ant Task Definitions

The table below contains the default task definitions included with the JWare/AntXtras Svn4Ant distribution. This table includes regular tasks as well as Svn4Ant macrodefs and presetdefs. To use these definitions, load the client or admin “`antlib.xml`” files into your Ant runtime as shown.

If the Svn4Ant jars and all of the required dependency jars are in your Ant runtime’s classpath, you can load Svn4Ant’s ***client*** components like:

```
<taskdef resource=”com/idaremedia/svn4ant/client/antlib.xml”/>
```

If the Svn4Ant jars and all of the required dependency jars are in your Ant runtime’s classpath, you can load Svn4Ant’s ***admin*** components like:

```
<taskdef resource=”com/idaremedia/svn4ant/admin/antlib.xml”/>
```

Category	Type Name	Class Name
Client	svn4ant	com.idaremedia.svn4ant.startup.Svn4AntBootstrapTask
	svn	com.idaremedia.svn4ant.client.jsvn.SvnCliTask
	svn-libcheck	com.idaremedia.svn4ant.client.info.SvnLibCheckTask
	svnrevget	com.idaremedia.svn4ant.client.solo.SvnGetRevisionTask
	svndirname	com.idaremedia.svn4ant.client.jsvn.SvnDirnameTask
	svnimport	com.idaremedia.svn4ant.client.solo.SvnImportTask
	svnexport	com.idaremedia.svn4ant.client.solo.SvnExportTask
	svncheckout	com.idaremedia.svn4ant.client.solo.SvnCheckoutTask
	svncat	com.idaremedia.svn4ant.client.solo.SvnCatTask
	svncommit	com.idaremedia.svn4ant.client.solo.SvnCommitTask
	svnupdate	com.idaremedia.svn4ant.client.solo.SvnUpdateTask
	svncopy	com.idaremedia.svn4ant.client.solo.SvnCopyTask
	svnbranch	com.idaremedia.svn4ant.client.branch.SvnBranchTask
	svnmove	com.idaremedia.svn4ant.client.solo.SvnMoveTask
	svnretire	com.idaremedia.svn4ant.client.branch.SvnRetireTask
	svndelete	com.idaremedia.svn4ant.client.solo.SvnDeleteTask
	svnadd	com.idaremedia.svn4ant.client.solo.SvnAddTask
	svnprop	com.idaremedia.svn4ant.client.solo.SvnPropTask
	svnrevert	com.idaremedia.svn4ant.client.solo.SvnRevertTask
Admin	svnadmin-libcheck	com.idaremedia.svn4ant.admin.SvnAdminLibCheckTask
	svnadmin-create	com.idaremedia.svn4ant.admin.SvnAdminCreateTask
	svnadmin-hotcopy	com.idaremedia.svn4ant.admin.SvnAdminHotCopyTask
	svnadmin-setlog	com.idaremedia.svn4ant.admin.SvnAdminSetLogTask
	svnadmin-dump	com.idaremedia.svn4ant.admin.SvnAdminDumpTask
	svnadmin-load	com.idaremedia.svn4ant.admin.SvnAdminLoadTask
	svadmin-verify	com.idaremedia.svn4ant.admin.SvnAdminVerifyTask

Svn4Ant Type Definitions

The table below contains the default type definitions included with the JWare / AntXtras Svn4Ant distribution. This table includes regular data types as well as Ant conditions and Ant fileset selector types. To use these definitions, load the client or admin "`antlib.xml`" files into your Ant runtime as shown.

If the Svn4Ant jars and all of the required dependency jars are in your Ant runtime's classpath, you can load Svn4Ant's ***client*** components like:

```
<taskdef resource="com/idaremedia/svn4ant/client/antlib.xml"/>
```

If the Svn4Ant jars and all of the required dependency jars are in your Ant runtime's classpath, you can load Svn4Ant's ***admin*** components like:

```
<taskdef resource="com/idaremedia/svn4ant/admin/antlib.xml"/>
```

Category	Type Name	Class Name
Client	svncredential	com.idaremedia.svn4ant.clientauth.SvnCredential
	svnserver	com.idaremedia.svn4ant.clientauth.SvnServerDef
	svnserverset	com.idaremedia.svn4ant.clientauth.SvnServerSet
	svntransferset	com.idaremedia.svn4ant.client.transfer.SvnTransferSet
	svnignorehandler	com.idaremedia.svn4ant.client.misc.SvnIgnoreHandler
	isversioned	com.idaremedia.svn4ant.client.misc.IsVersioned
	inversioned	com.idaremedia.svn4ant.client.misc.InVersionedDirectory
	isversionedroot	com.idaremedia.svn4ant.client.misc.IsWorkingCopyRoot
Admin	isrepository	com.idaremedia.svn4ant.admin.IsRepository

Svn4Ant Value URI Handler Definitions

The table below contains the builtin Svn4Ant value URI handlers and their default schema names for the JWare/AntXtras Svn4Ant distribution. To use these value URIs you need to define and enable them with the standard AntXtras `<manageuris>` task as shown below.

```
<manageuris action="enable">
    <parameter name="svnurl"
    value="com.idaremedia.svn4ant.client.misc.RepositoryUrlValueURIHandler"/>
    <parameter name="svndir"
    value="com.idaremedia.svn4ant.client.misc.RepositoryDirValueURIHandler"/>
    <parameter name="loadfile"
        value="com.idaremedia.svn4ant.valueuri.LoadFileValueURIHandler"/>
    <parameter name="os"
        value="com.idaremedia.svn4ant.valueuri.IsOsValueURIHandler"/>
</manageuris>
```

Schema Name	Class Name
`$svnurl:`	`com.idaremedia.svn4ant.client.misc.RepositoryUrlValueURIHandler`
`$svndir:`	`com.idaremedia.svn4ant.client.misc.RepositoryDirValueURIHandler`
`$loadfile:`	`com.idaremedia.svn4ant.valueuri.LoadFileValueURIHandler`
`$os:`	`com.idaremedia.svn4ant.valueuri.IsOsValueURIHandler`

GNU Free Documentation License

Version 1.2, November 2002